To apply the scientific method to develop and validate a hypothesis about the order of growth of the running time of the given code fragment in Exercise 4.1.13, we will follow these steps:

1. **Observation**:
   - We observe the code fragment to understand its behavior and identify potential patterns.
2. **Formulate a Hypothesis**:
   - Based on the structure and behavior of the code, we will hypothesize the order of growth of its running time as a function of n.
3. **Predict**:
   - Using our hypothesis, we will predict the running time for specific values of n.
4. **Experiment**:
   - We will measure the actual running time for various values of n to test our hypothesis.
5. **Analyze Data**:
   - We will compare the predicted and actual running times to validate or refine our hypothesis.
6. **Conclude**:
   - Based on the analysis, we will draw conclusions about the running time of the code.

Step-by-Step Analysis

1. **Observation**:
   - The code fragment defines a recursive function f that calls itself twice for each invocation, except when n is 0.
   - The base case returns 1 immediately.

```
public static int f(int n)
{
    if (n == 0) return 1;
    return f(n-1) + f(n-1);
}
```

2. **Formulate a Hypothesis**:
   - Each call to $f(n)$ results in two calls to $f(n-1)$. This indicates a binary branching recursion.
   - The recurrence relation for the number of calls can be written as: $T(n) = 2T(n-1) + O(1)$

- Solving this recurrence relation suggests exponential growth. Specifically, we hypothesize that $T(n)$ is $O(2^n)$.

3. **Predict**:

- If our hypothesis is correct, the number of calls (and thus the running time) should double with each increment of n.
- For small values:

$$T(0) = 1$$

$$T(1) = 2 \cdot T(0) = 2$$

$$T(2) = 2 \cdot T(1) = 4$$

$$T(3) = 2 \cdot T(2) = 8$$

Generally, $T(n) = 2^n$.

4. **Experiment**:

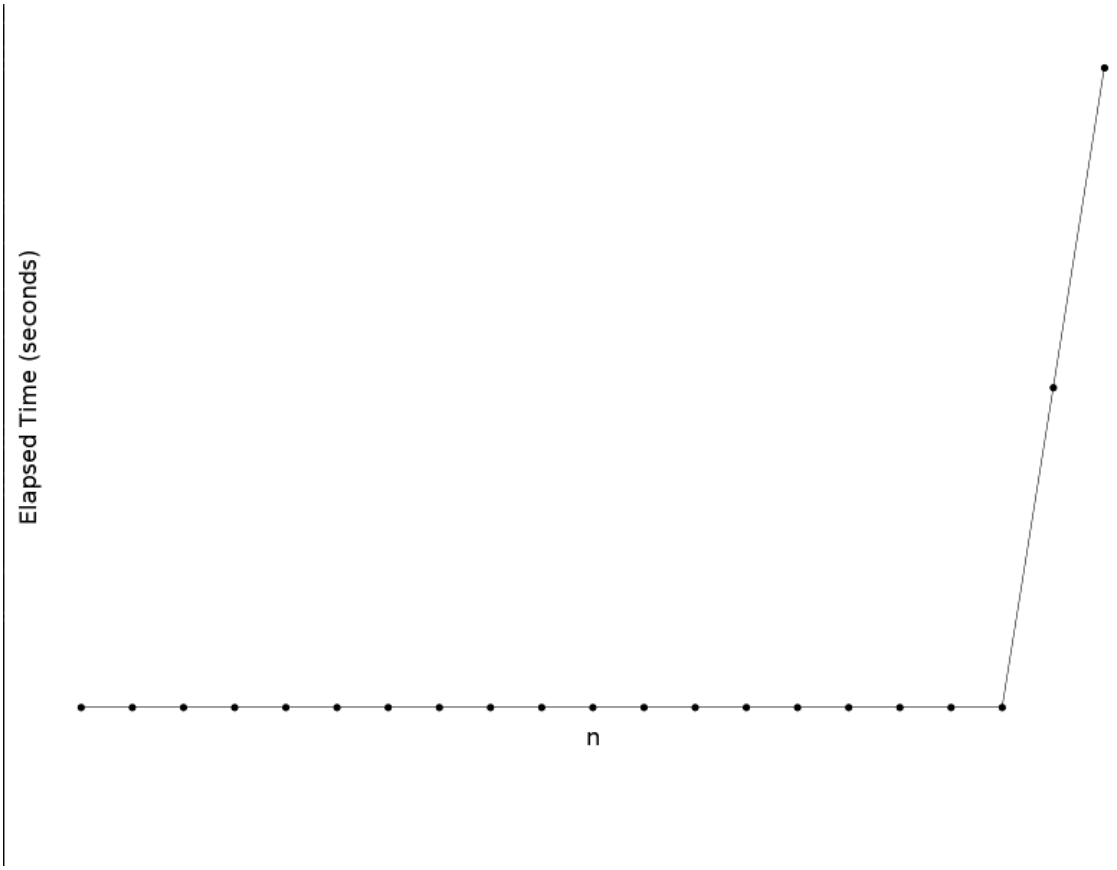
- Implement timing code to measure the running time for various values of n.

```
public static void main(String[] args) {
   int maxN = 20;
   double[] times = new double[maxN + 1];
   for (int i = 0; i <= 20; i++) {
   Stopwatch timer = new Stopwatch();
   f(i);
   double elapsedTime = timer.elapsedTime();
   times[i] = elapsedTime;
   StdOut.println("n=" + i + " elapsed time = " +
elapsedTime);
    }
}
```

5. **Analyze Data**:

- Collect the running times and plot them on a graph or compare the ratios of successive running times.
- If the hypothesis is correct, we expect the running time to approximately double for each increment of n.

```
// Plot points
 for (int i = 0; i <= maxN; i++) {
     StdDraw.point(i, times[i]);
 }
```



Conclusion

- **Hypothesis**: The running time of the code fragment is $O(2^n)$.
- **Validation**: If experimental data shows exponential growth (doubling for each increment of n), the hypothesis is validated.
- If the data deviates, analyze potential causes (e.g., system overhead, measurement inaccuracies) and refine the hypothesis accordingly.

By following these steps, we can systematically analyze and validate the order of growth of the running time for the given code fragment.